

Posaonova jendačina

Sadržaj

Postavka problema.....	1
Formiranje mreže.....	2
Formiranje matrice sistema.....	3
Retke matrice u Matlabu.....	5
Desna strana sistema.....	7
Rešenje	8
Grafik.....	9
Greška.....	13
Konvergencij.....	13

Postavka problema

Aproksimiramo granični problem

$$\begin{aligned}-\Delta u(x, y) &= f(x, y), \quad (x, y) \in \Omega, \\ u(x, y) &= g(x, y), \quad (x, y) \in \Gamma = \partial\Omega,\end{aligned}$$

metodom konačnih razlika. Pretpostavimo da je Ω pravougaona oblast

$$\Omega = (x_{min}, x_{max}) \times (y_{min}, y_{max})$$

i jednostavnosti radi izabraćemo da koraci po x i y osi budu jednaki $h = h_x = h_y$.

Metodu ćemo implementirati za homogene Dirihleove granične uslove, a za prvi test primer izabraćemo problem koji se tačno aproksimira. Konačna razlika reda dva će sigurno aproksimirati tačno polinom stepena dva. Ako je

$$\bar{\Omega} = [0, 3] \times [0, 4],$$

tada rešenje možemo izabrati kao

$$u(x, y) = x(3 - x)y(4 - y).$$

Kako je

$$\frac{\partial^2 u}{\partial x^2} = -2y(4 - y)$$

i

$$\frac{\partial^2 u}{\partial y^2} = -2x(3 - x)$$

to je desna strana sistema jednaka

$$f(x, y) = 2x(3 - x) + 2y(4 - y).$$

Da bismo rešili problem, prvo formiramo ravnomerne mreže:

$$\bar{\omega}_{hx} = \{x_i \mid x_i = x_0 + ih_x, x_0 = xmin, h_x = (xmax - xmin)/n, n > 1\},$$

$$\bar{\omega}_{hy} = \{y_j \mid y_j = y_0 + jh_y, y_0 = ymin, h_y = (ymax - ymin)/m, m > 1\}$$

i $\bar{\omega}_h = \bar{\omega}_{hx} \times \bar{\omega}_{hy}$. Zatim u unutrašnjim tačkama mreže aproksimacija glasi

$$-v_{xx,ij} - v_{yy,ij} = f, i = 1, 2, \dots, n-1, j = 1, \dots, m-1,$$

a na granici

$$v_{0j} = g_{0j}, \quad v_{nj} = g_{nj}, \quad j = 0, 1, \dots, m,$$

$$v_{i0} = g_{i0}, \quad v_{im} = g_{im}, \quad i = 0, 1, \dots, n.$$

Formiranje mreže

Sve podatke o problemu domen, desnu stranu jednačine, eventualno tačno rešenje ćemo sačuvati u posebnom skript fajlu. Ovde su to *PrimerP1* koji smo izveli gore i još dva primera, na kojima ćemo testirati konvergenciju metode.

Da bismo formirali mrežu, prvo učitavamo podatke sa

```
primer='primerP1';
run(primer)
```

Radi lakšeg analiziranja koda, *PrimerP1* je izabran tako da čvorovi mreže imaju iste vrednosti kao i njihovi indeksi. Dakle, u ovom test primeru važi da je

$$x_i = i$$

$$y_j = j,$$

ako je $h = h_x = h_y = 1$. Mrežu po x i y osi, respektivno, možemo formirati sa

```
h=1;
xosa=xmin:h:xmax;
yosa=ymin:h:ymax;
nx=length(xosa);
ny=length(yosa);
```

Kako imamo 2D problem, neophodan nam je Dekartov proizvod ovih tačaka, zapravo treba da formiramo mrežu. Najčešće se koristi za 2D plotovanje *meshgrid*. Ako želimo 2D mrežu, kao ulazne argumente unosimo vektore sa x i y ose, redom, a kao rezultat dobijamo dve matrice X i Y

```
[X,Y]=meshgrid(xosa,yosa)
```

```
X = 5x4
    0    1    2    3
    0    1    2    3
    0    1    2    3
    0    1    2    3
    0    1    2    3
Y = 5x4
    0    0    0    0
```

1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4

Jedna tačka mreže je $(x_i, y_j) = (xosa(i), yosa(j))$ i pristupamo joj sa

$$X(j, i) = xosa(i),$$

$$Y(j, i) = yosa(j).$$

Ovo nije u saglasnosti sa indeksiranjem matrica i morali bismo sa većom pažnjom pisati program.

Jednostavnije i smislenije bi bilo da pristupamo sa

$$X(i, j) = xosa(i),$$

$$Y(i, j) = yosa(j).$$

Upravo zbog toga je u Matlabu implementirana i druga metoda *ndgrid* koja se poziva analogno kao i *meshgrid*, ali formira matrice *X* i *Y* u skladu sa indeksiranjem matrica:

```
[X,Y]=ndgrid(xosa,yosa)
```

X = 4x5

0	0	0	0	0
1	1	1	1	1
2	2	2	2	2
3	3	3	3	3

Y = 4x5

0	1	2	3	4
0	1	2	3	4
0	1	2	3	4
0	1	2	3	4

Formiranje matrice sistema

Neka je $h = h_x = h_y = 1$. Aproksimacijom našeg problema metodom konačnih razlika dobijamo sledeći sistem jednačina u unutrašnjim tačkama (krst shema):

$$4v_{11} - v_{10} - v_{12} - v_{01} - v_{21} = h^2 f_{11},$$

$$4v_{12} - v_{11} - v_{13} - v_{02} - v_{22} = h^2 f_{12},$$

$$4v_{13} - v_{12} - v_{14} - v_{03} - v_{23} = h^2 f_{13},$$

$$4v_{21} - v_{20} - v_{22} - v_{11} - v_{31} = h^2 f_{21},$$

$$4v_{22} - v_{21} - v_{23} - v_{12} - v_{32} = h^2 f_{22},$$

$$4v_{23} - v_{22} - v_{24} - v_{13} - v_{33} = h^2 f_{23},$$

i na granici

$$v_{0,j} = g_{0,j}, \quad j = 0, 1, \dots, m$$

$$v_{n,j} = g_{n,j}, \quad j = 0, 1, \dots, m$$

$$v_{i,0} = g_{i,0}, \quad i = 0, 1, \dots, n$$

$$v_{i,m} = g_{i,m}, \quad i = 0, 1, \dots, nx.$$

Sve nepoznate elemente $v_{i,j}$ smeštamo u vektor kolonu na sledeći način

$$v = [v_{00}, v_{01}, \dots, v_{0,m}, v_{10}, \dots, v_{1,m}, \dots, v_{n,0}, \dots, v_{n,n}]^T,$$

pa je odgovarajući sistem jednačina

[illegible]

gde na praznim mestima treba da stoje nule. Očigledno, ovo je retka, tačnije petodijagonalna matrica.

U slučaju da su zadati homogeni Dirihleovi granični uslovi, kao u našem primeru, elemente na granici možemo izbaciti iz sistema, pa se time on svodi na:

$$\left[\begin{array}{ccc|ccc} 4 & -1 & & -1 & & \\ -1 & 4 & -1 & & -1 & \\ & -1 & 4 & & & -1 \\ - & - & - & - & - & - \\ -1 & & & 4 & -1 & \\ & -1 & & -1 & 4 & -1 \\ & & -1 & & -1 & 4 \end{array} \right] \begin{bmatrix} v_{11} \\ v_{12} \\ v_{13} \\ - \\ v_{21} \\ v_{22} \\ v_{23} \end{bmatrix} = h^2 \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ - \\ f_{21} \\ f_{22} \\ f_{23} \end{bmatrix}.$$

Dobili smo daleko manju matricu, čim ćemo uštedeti radnu memoriju, pa ćemo moći rešavati i probleme većih dimenzija. Dimenzija matrice sistema je sada $[(nx - 2)(ny - 2)]^2$. Jednostavnosti radi, uvešćemo promenljive

```
Nx=nx-2;
Ny=ny-2;
```

Neka je A matrica skraćenog sistema. Primetimo sledeće:

- na dijagonali matrice svi elementi su jednaki 4, dužina odgovarajućeg vektora je $N_x N_y$
- na prvoj naddijagonali i prvoj poddijagonali su elementi $[-1, -1, 0, -1, -1]$, dužine $N_x N_y - 1$; primetimo da je deo $[-1, -1, 0]$ dužine N_y i da će se ponavljati N_x puta (poslednji će biti kraći za jedan poslednji element)
- na sledećoj nenula naddijagonali (i poddijagonali) je vektor $[-1, -1, -1]$; on se nalazi na poziciji udaljenoj od glavne dijagonale za N_y mesta, pa je njegova dužina $N_x N_y - N_y = N_y(N_x - 1)$

Ove osobine ćemo iskoristiti da formiramo odgovarajuće vektore:

```
% glavna dijagonala
d0=4*ones(Nx*Ny,1);
%prva naddijagonala i poddijagonala
d1=-1*ones(Ny,1);
d1(Ny)=0;
d1= repmat(d1, Nx,1); % repmat funkcija za nadovezivanje vektora
%sledeca nenula poddijagonala i naddijagonala
d2=-1*ones(Ny*(Nx-1),1);
```

Sada matricu možemo formirati koristeći Matlabovu funkciju *diag*:

```
A=diag(d0)+diag(d1(1:Nx*Ny-1),1)+diag(d1(1:Nx*Ny-1),-1)+diag(d2,Ny)
+diag(d2,-Ny); % ovde skratimo vektor vektor d1
disp(A)
```

```

4    -1     0    -1     0     0
-1     4    -1     0    -1     0
0     -1     4     0     0    -1
-1     0     0     4    -1     0
0     -1     0    -1     4    -1
0     0    -1     0    -1     4
```

Retke matrice u Matlabu

Očigledno, ako budemo formirali punu matricu, vrlo brzo ćemo ostati bez radne memorije, jer sistemi mogu biti prilično velikih dimenzija. Zato je bolje matricu tretirati kao retku. U matlabu postoje funkcije za rad sa retkim matricama. Da bismo od vektora dijagonala formirali retku matricu koristimo Matlabovu funkciju *spdiags*. Unosimo sledeće argumente:

- pomoćnu matricu koja kao kolone sadrži samo elemente sa nenula dijagonala
- vektor vrstu koji sadrži brojeve dijagonala na koje kolone pomoćne matrice treba da se postave
- broj vrsta matrice koja treba da se formira
- broj kolona matrice koja treba da se formira.

Svi vektori koji idu na dijagonale moraju biti iste dužine kao i glavna dijagonala. S obzirom da prosleđujemo i broj dijagonale k , Matlab onda odseca vektore koji idu na dijagonale na sledeći način, ako je broj vrsta veći ili jednak od broja kolona:

- iz vektora poddijagonala poslednjih k elemenata
- iz vektora naddijagonala prvih k elemenata

U skladu sa tim vektore koji idu na poddijagonale i naddijagonale možemo dopuniti bilo kojim vrednostima. Pa će biti

```
d1d=d1; % prva poddijagonala: d1 je vec duzine Nx*Ny, poslednji element je
0, koji i ne ide u matricu, matlab će odseći tu vrednost
d1u=[0;d1(1:end-1)];% prva naddijagonala, ovde odseca prvi element, pa
pomeramo sve vrednosti za jedno mesto udesno, izbacujemo poslednji element,
a na prvo mesto stavljamo 0
d2d=[d2; zeros(Ny,1)];% sledeca nenula poddijagonala, odseci ce poslednjih
Ny elemenata
d2u=[zeros(Ny,1);d2];% sledeca nenula naddijagonala, odseci ce prvih Ny
elemenata
D=[d2d,d1d,d0,d1u,d2u];
disp(D)
```

-1	-1	4	0	0
-1	-1	4	-1	0
-1	0	4	-1	0
0	-1	4	0	-1
0	-1	4	-1	-1
0	0	4	-1	-1

```
k=[-Ny,-1,0,1,Ny];
A=spdiags(D,k,Nx*Ny,Nx*Ny)
```

```
A = 6x6 sparse double matrix (20 nonzeros)
(1,1)      4
(2,1)     -1
(4,1)     -1
(1,2)     -1
(2,2)      4
(3,2)     -1
(5,2)     -1
(2,3)     -1
(3,3)      4
(6,3)     -1
```

(1,4)	-1
(4,4)	4
(5,4)	-1
(2,5)	-1
(4,5)	-1
(5,5)	4
(6,5)	-1
(3,6)	-1
(5,6)	-1
(6,6)	4

Pogledati dokumentaciju funkcije `spdiags`. Kao rezultat vraća strukturu podataka koji čuva samo nenula elemente i njihove indekse u matrici.

Ako želimo da formiramo punu matricu, koristimo naredbu `full(A)`:

```
full(A)
```

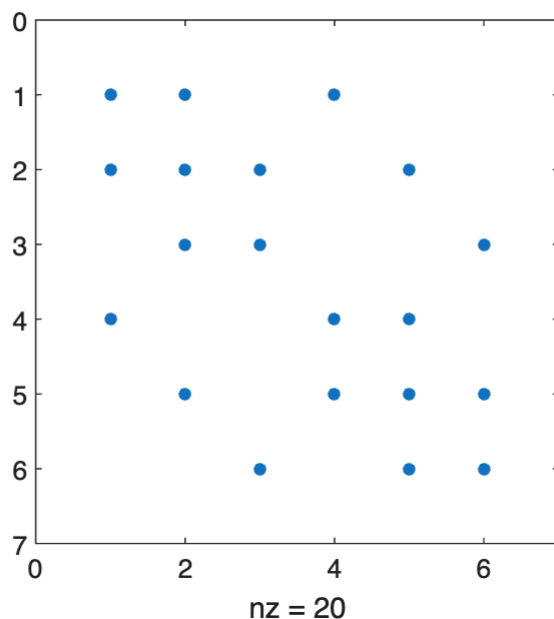
```
ans = 6x6
```

```

4    -1    0    -1    0    0
-1    4    -1    0    -1    0
0    -1    4    0    0    -1
-1    0    0    4    -1    0
0    -1    0    -1    4    -1
0    0    -1    0    -1    4
```

Ako želimo samo grafički prikaz strukture matrice, koristim naredbu `spy(A)`

```
spy(A)
```



Sada smo spremni da formiramo i desnu stranu sistema.

Desna strana sistema

S obzirom na to da imamo homogene Dirihleove granične uslove, dovoljno je da izračunamo vrednosti desne strane samo u unutrašnjim čvorovima mreže:

```
F=h^2*f(X(2:nx-1,2:ny-1),Y(2:nx-1,2:ny-1));
```

Ovim smo dobili matricu oblika

$$F = h^2 \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \end{bmatrix}.$$

U Matlabu se sve matrice čuvaju kao vektori vrste, tako što se kolone matrice slažu redom u vektor, pa komandom

```
F1d=F(1:Nx*Ny)
```

```
F1d = 1x6
    10    10    12    12    10    10
```

dobijamo vektor vrstu

$$F1d = h^2 [f_{11} \ f_{21}, \ f_{12}, \ f_{22}, \ f_{13}, \ f_{23}].$$

Ovaj raspored nije tačan za naš sistem jednačina u slučaju da matrica F ima više od jedne vrste, ali transponovanjem matrice F

```
if Nx>1
    F=F';
end
```

$$F = h^2 \begin{bmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \\ f_{13} & f_{23} \end{bmatrix}.$$

i zatim prevođenjem u vektor

```
F1d=F(1:Nx*Ny)
```

```
F1d = 1x6
    10    12    10    10    12    10
```

dobijamo dobar raspored elemenata.

Rešenje

Rešenje sistema lako dobijamo koristeći matlabov ugrađeni operator $\backslash$, gde desnu stranu sistema $F1d$ treba transformisati u vektor kolonu:

```
v=A\F1d'
```

```
v = 6x1
    6.0000
    8.0000
```

```
6.0000
6.0000
8.0000
6.0000
```

Rešenje je koje dobijamo je složeno u vektor kolonu

$$v = [v_{11}, v_{12}, v_{1,N_y}, v_{21}, \dots, v_{2,N_y}, \dots, v_{N_x,1}, \dots, v_{N_x,N_y}]^T,$$

pa je za dalji rad potrebno konvertovati ovaj vektor u matricu. Za to koristimo Matlabovu funkciju *reshape*. Kao argumente zadajemo vektor koji treba konvertovati, a zatim dimenziju matrice u koju treba da se konvertuje vektor.

Ako je kao argument data dimenzija $[n, m]$, Matlab će uzimati redom po n elemenata i smeštati ih u kolone (ukupno njih m). Stoga, u našem zadatku je potrebno zadati dimenziju $[N_y, N_x]$, a zatim transponovati dobijenu matricu

```
V=reshape(v, [Ny,Nx])';
disp(V)
```

```
6.0000    8.0000    6.0000
6.0000    8.0000    6.0000
```

Na kraju, ako želimo dodati i vrednosti na granici, potrebno je proširiti matricu V sa nulama na sledeći način:

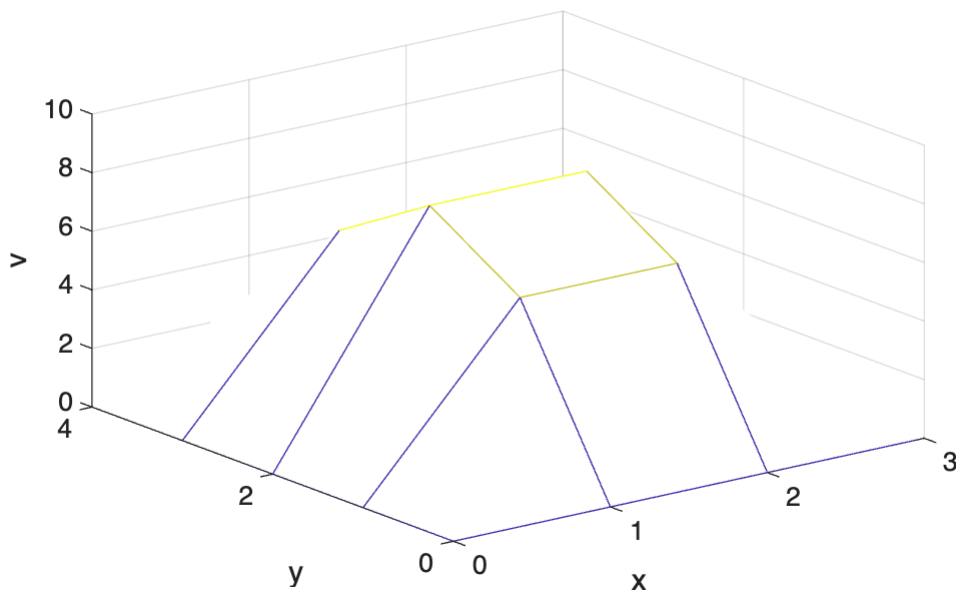
```
V=[zeros(nx,1), [zeros(1,Ny);V;zeros(1,Ny)], zeros(nx,1)];
disp(V)
```

```
0         0         0         0         0
0    6.0000    8.0000    6.0000    0
0    6.0000    8.0000    6.0000    0
0         0         0         0         0
```

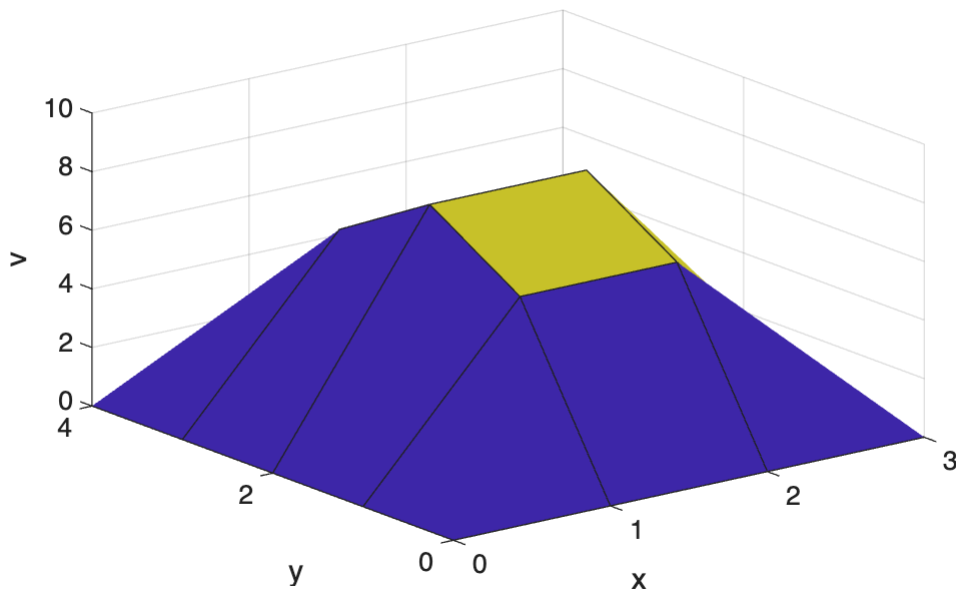
Grafik

Grafički prikaz rešenja se sada lako dobija funkcijama *mesh*, *surf* ili *plot3*.

```
mesh(X,Y,V)
xlabel('x')
ylabel('y')
zlabel('v')
```



```
surf(X,Y,V)
xlabel('x')
ylabel('y')
zlabel('v')
```



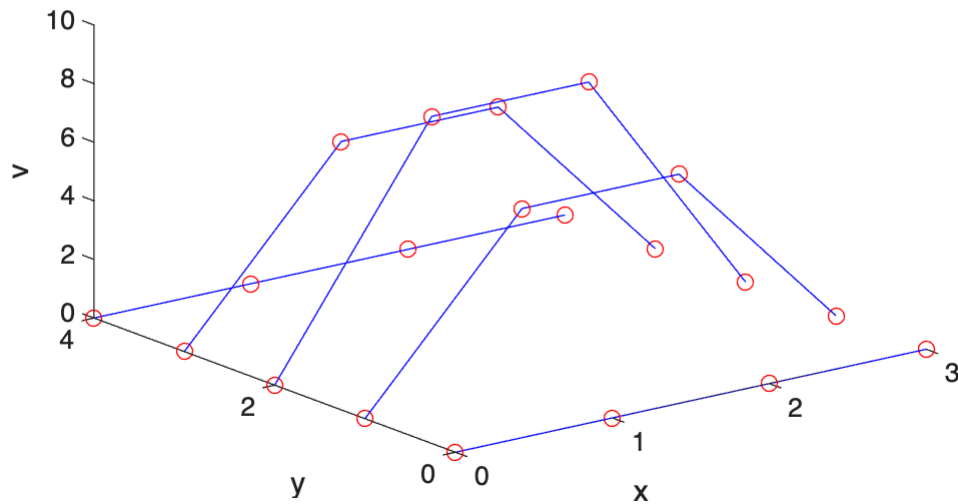
Od ove tri funkcije, možda je izbor *plot3* najkorisniji sa stanovišta numerike, jer ako na grafiku prikažemo i tačno rešenje, možemo videti poklapanja/odstupanja aproksimacije od tačnog rešenja. Mada, da bi se videlo ponašanje sheme, sasvim je dovoljno prikazati samo 2D grafik za jedno fiksirano x ili y .

```
plot3(X,Y,V,'ro-')
hold on
plot3(X,Y,ue(X,Y),'b')
```

```

hold off
xlabel('x')
ylabel('y')
zlabel('v')

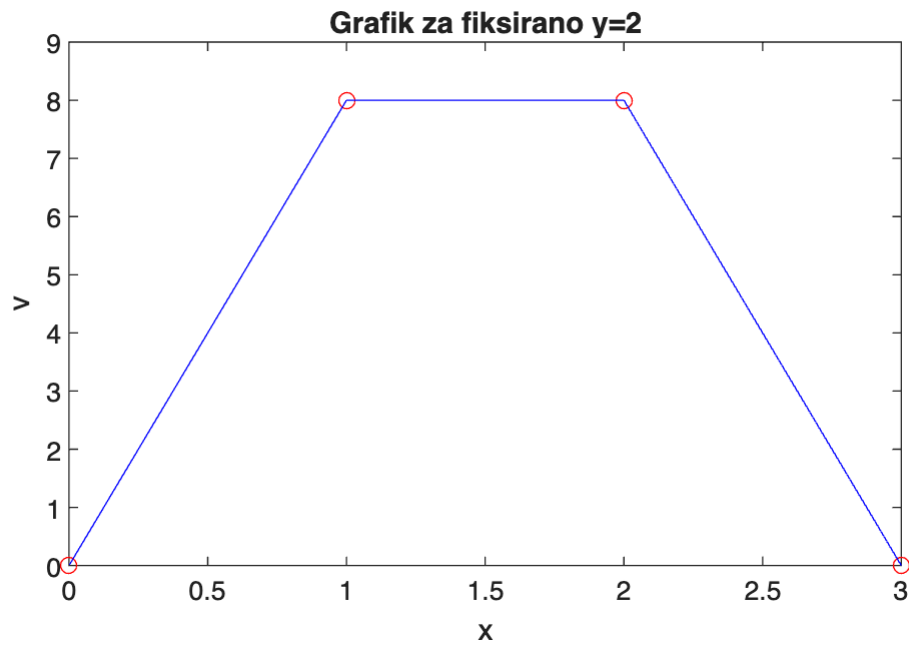
```



```

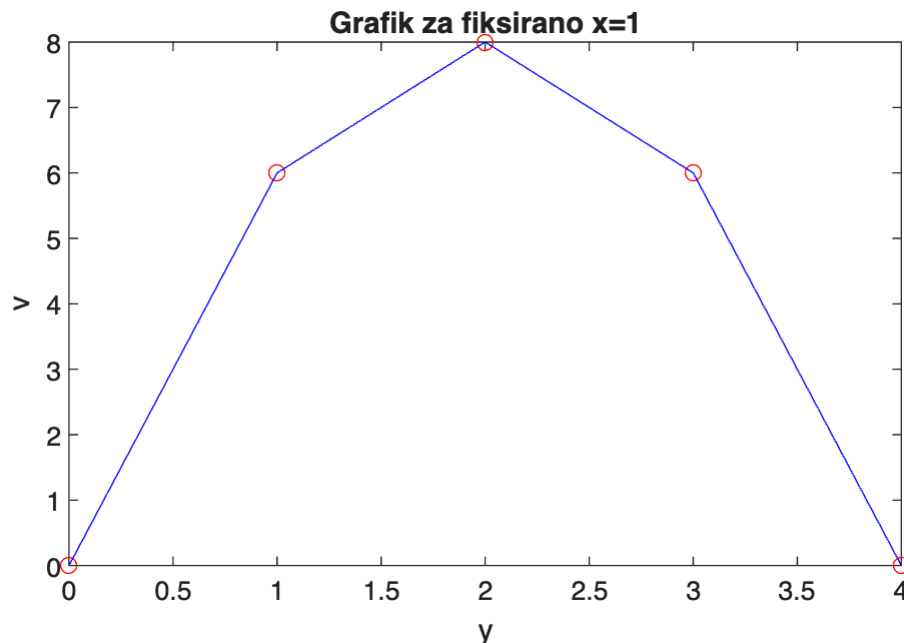
if mod(ny,2)==0
    jg=ny/2;%na primer ako zelimo prikazati grafik za ovo fiksirano y
else
    jg=(ny+1)/2;
end
plot(xosa,V(:,jg),'ro-')
hold on
yg=yosa(jg);
plot(xosa,ue(xosa,yg),'b')
hold off
xlabel('x')
ylabel('v')
title("Grafik za fiksirano y="+yg)

```



yg =
2

```
if mod(nx,2)==0
    ig=nx/2;%na primer ako zelimo prikazati krafik za ovo fiksirano x
else
    ig=(nx+1)/2;
end
plot(yosa,V(ig,:), 'ro-')
hold on
xg=xosa(ig);
plot(yosa,ue(xg,yosa), 'b')
hold off
xlabel('y')
ylabel('v')
title("Grafik za fiksirano x="+xg)
```



yg =
2

Greška

I na kraju, odredimo uniformnu normu greške aproksimacije

```
U=ue(X,Y);%tacno resenje
errm=max(max(abs(U-V)));
```

Konvergencij

```
red=redkonv(@mkrP,'primerP1')%zadatak se tacno aproksimira, jedina greska
je racunska, pa je red konvergencije nepravilan
```

```
red = 1x7
    -0.4854    -3.0000    -2.4021    -1.5620    -2.0661    -1.9429    -1.9116
```

```
red=redkonv(@mkrP,'primerP2')%ovde su umesto polinoma trigonometrijske
funkcije
```

```
red = 1x7
    2.1398    2.0338    2.0084    2.0021    2.0005    2.0001    2.0000
```

```
red=redkonv(@mkrP,'primerP3')%racionalne
```

```
red = 1x7
    1.9817    1.9921    1.9867    1.9987    1.9990    1.9999    1.9999
```

```
type redkonv
```

```
function red=redkonv(fun,primer)
% Funkcija red=redkonv(primer) odredjuje red konvergencije
% metode konacne razlike primenjene na Puasonovu jednacinu
% sa homogenim granicnim uslovima, tj. funkcija fdmP()
```

```

%
% Test: red=redkonv('primerP1') %Ovo je jednostavan primer za
% koji numericka metoda ne pravi gresku. Zbog toga se ukupna greska
% sastoji samo od racunske, pa red konvergencije ne moze biti 2.
%
%       red=redkonv('primerP2')
%       red=redkonv('primerP3')
%
% Ulazni podaci:
%       fun - pokazivac na funkciju
%       primer - ime skript fajla u kome se nalaze:
%               xmin, xmax, ymin, ymax, f, ye tacno resenje
% Izlazni podaci:
%       red - vektor, red konvergencije izracunavamo
%             poloveci korak h vise puta

k=8;% najvisi stepen za h=1/2
m=1:k;% korsiticemo stepene od 1 do k
h=1./(2.^m);% h je vektor svih koraka koje cemo koristiti za
%odredjivanje reda konv
e=zeros(1,k);

for i=1:k
    e(i)=fun(primer,h(i));%odredjujemo gresku
end

%red konvergencije: log2(e_h/e_(h/2))
red=log2(e(1:k-1)./e(2:k));

```